THE HAZARD NOTIFICATION SYSTEM (HANS)

BY

SETH FRANK SNEDIGAR

B.S., Montana Tech of the University of Montana, 2001

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Department of Computer Science of the
University of Alaska, Fairbanks, 2009

Fairbanks, Alaska

TABLE OF CONTENTS

INTRODUCTION

This paper describes the Hazard Notification System (HANS), a networked system for distributing volcanic activity information collected by scientists to pilots, emergency services, and the general public.

There are five officially sanctioned volcano observatories in the United States – the Alaska Volcano Observatory (AVO), with offices in Fairbanks, AK and Anchorage, AK, the Cascades Volcano Observatory (CVO), based out of Vancouver, WA, the Hawaiian Volcano Observatory (HVO), headquartered near Volcano, HI, the Long Valley Observatory (LVO), based in Menlo Park, CA, and finally the Yellowstone Volcano Observatory (YVO) based in Menlo Park, CA and Salt Lake City, UT. Heading up all these observatories is the Volcano Hazards Program (VHP), a United States Geological Survey (USGS) program based in Reston, VA.

The VHP monitors active and potentially active volcanoes, assesses their hazards, responds to volcanic crises, and conducts research on how volcanoes work to fulfill Congressional mandate P.L. 93-288 – that the USGS issue "timely warnings" of potential volcanic hazards to responsible emergency-management authorities and to the populace affected [11].

Each observatory constantly releases official volcanic information to the general public and interested parties, as part of fulfilling the congressional mandate. This information is released in a specifically formatted text document, the contents of which are emailed to various subscription lists, faxed to dozens of fax machines, and posted to many websites throughout the internet.

While all observatories held to a somewhat common formatting model, no two information products from any particular observatory were very similar. Also, the method of storing and cataloging these released products varied greatly among the observatories. In late 2005, a variety of events initiated ~~talk~~discussions of a system which could help all observatories release their information in a unified format, and store it in an easily retrievable and searchable manner. This was the beginning of the Hazard Notification System (HANS). As a matter of simplification, the HANS will henceforth be referred to as simply *HANS*, rather than *the HANS*.

The AVO was selected to develop this system, as they have the necessary resources to put towards a project like this, as well as plenty of volcanic activity on which to test the various information products that would be created and disseminated by HANS.

PREVIOUS VERSIONS

One of the reasons AVO was selected to lead the development of HANS was that they had developed and used several prior versions of a HANS-type system. We call these versions v0.1, v1.0, and v2.0.

HANS v0.1

HANS v0.1 could barely be considered a system. An AVO scientist would open up the Microsoft Word application, and begin creating the information release by hand. Once they had crafted the report, they copied the text into an email client and manually sent out the email to a distribution list. Then they printed out the document and faxed the document using an online fax distribution service. Finally, they called the webmaster of the AVO website, and the webmaster would update the appropriate web pages with the latest information.

This method was very prone to error, as the scientists would often just reuse the last information release document, but would forget to change volcano names, timestamps, etc. It was also fairly labor intensive, especially during times of heightened volcanic activity, when several releases were often put out per day, and the webmaster would be required to make constant changes to the website.

HANS v1.0

In 2004, AVO hired an Analyst/Programmer, the author, to take over the development and maintenance of the AVO public website (http://www.avo.alaska.edu). One of the first things developed by this programmer could be considered the first version of HANS. This version still required a scientist to create the original document in Word, or some other text editor, but then the scientist could paste the text into a web form, submit the form, and the system would update a database, which would in turn update the necessary web pages, as well as send out appropriate emails.

While this system did not solve the formatting issues, it did help remedy some of the labor intensive portions required of releasing updates, primarily the constant need to manually update the website.

HANS v2.0

In mid 2005, a requirement was given to all volcano observatories that they would need to release their information formatted to fit the Common Alerting Protocol (CAP). CAP is an XML based data format for exchanging public warnings and emergencies between alerting technologies [2]. CAP messages are formatted into several fields within the XML document, and are transmitted to a national database storing disaster information. This national system is called the Disaster Management Interoperability Services (DMIS).

AVO volunteered to lead the development of a system that would fulfill this requirement, as well as test the new release format for the other observatories. Since the CAP format required each message to be broken up into multiple fields, the new system required information releases to be broken up into different sections to fit those fields in the CAP message. However, the observatory still wanted to release information in the original formats using the same text inputted. As such, a requirement of the new system was to have one data entry form, and multiple product outputs from that form.

AVO created the database structure and web system to allow users to enter their information in one place, and submit both the original release and the associated CAP formatted textual output. While this system served the purpose of creating multiple products, transmitting data to the DMIS servers, and solving the formatting issues mentioned earlier, it proved fairly difficult to use, and quite unwieldy when trying to export the software to work for different observatories.

ADDITIONAL FACTORS FOR A NEW SYSTEM

Since their inception, the US volcano observatories have generally used a color code system – GREEN, YELLOW, ORANGE, and RED – to assign a code to the level of activity occurring at a particular volcano. However, the criteria for assigning a particular color code to a volcano focused primarily on the hazards to aviation around the volcano. In 2006, the USGS moved from using the term "color code" to the term "aviation color code", and added a new classification system called the volcano alert level [3]. The volcano alert level uses a four-tier system – NORMAL, ADVISORY, WATCH, and WARNING – to classify a volcano's activity with respect to hazards that may impact people on the ground. Thus, activity at a volcano is classified regarding hazards to aviation and hazards to people on the ground. HANS v2.0 was modified to implement this change to the volcano activity classification system. The modification was only a slight change, but still fairly difficult to implement.

The final straw for v2.0 came in 2007 when the International Civil Aviation Organization (ICAO) working with VHP personnel developed a standardized template for releasing volcano information to the aviation community. This template, called the Volcano Observatory Notice for ~~Notification~~Aviation is formatted exactly the same between observatories, and outlines current volcanic activity hazards that relate to aviation [4]. Complementing the VONA, is another template: the Volcano Activity Notification, or VAN. The VAN is similar to the VONA, different only that it is designed to convey information about volcanic hazards to those people on the ground. These products are expected to be generated from the same information (so the user only has to enter information once), then sent off to separate locations, as each product is formatted specifically for its appropriate industry.

HANS v2.0 could have been modified to accommodate these two new reports, but at this point it was painfully obvious that the current system was inflexible, and a ground-up rewrite would be much easier and would return much better results than attempting to modify the existing system.

v3.0 REQUIREMENTS & DESIGN

This project involved the design and implementation of HANS v3.0. HANS v3.0 is designed primarily to be flexible. As has been seen before, new products may be required of the observatories quite frequently. Thus, the system needs to be able to easily accommodate new formatted products, while remaining as user friendly as possible. The system should allow scientists to choose the appropriate product, enter the information necessary for that product, then release that information through the appropriate channels.

The system will also archive all released products in a searchable database. This database can be searched by both scientists internal to the observatories as well as the general public. Finally, the system will be fault tolerant – information will be synchronized across several systems.

This fault tolerant capability will be useful for several reasons. The primary reason is due to network failures. These network failures may occur for two reasons – failure due to hardware issues, and failure due to server overload. The second issue has already occurred at AVO during the early 2009 eruption of Redoubt volcano. High levels of earthquake activity prompted AVO to raise the color code and alert level of Redoubt; news of this heightened activity spread nationwide, and directed a flood of traffic towards the AVO webserver, which promptly slowed to an unusable crawl. AVO was able to use a HANS instance running on an unaffected server to keep releasing information about the volcano through the proper channels. (Of course, this crash may have been prevented through more robust server systems, which will hopefully be deployed in the near future.)

Also, as data is synchronized across all HANS instances, this can be considered a simple backup scheme. In the case that a system should suffer a massive disk failure, with no working backups, all released information for all observatories will still be recoverable from the other instances.

To accommodate all these requirements, HANS has been logically designed as four separate modules.

Data Entry Module

The data entry module interacts with the user as they enter their data into the database. The user is able to choose the observatory their report will be for, the type of report they want to send out, and the volcano or volcanoes the report will cover. Once these early parameters are chosen by the user, they are then taken to a data entry page with the necessary form fields to complete the type of report they have chosen. This data entry page is pre-filled in with the data that was sent out in the last release. Thus, if a user is entering daily updates about a specific volcano, the information pre-filled in will be from the day before. This can greatly reduce the amount of typing the user has to do, particularly if not much has changed at the volcano.

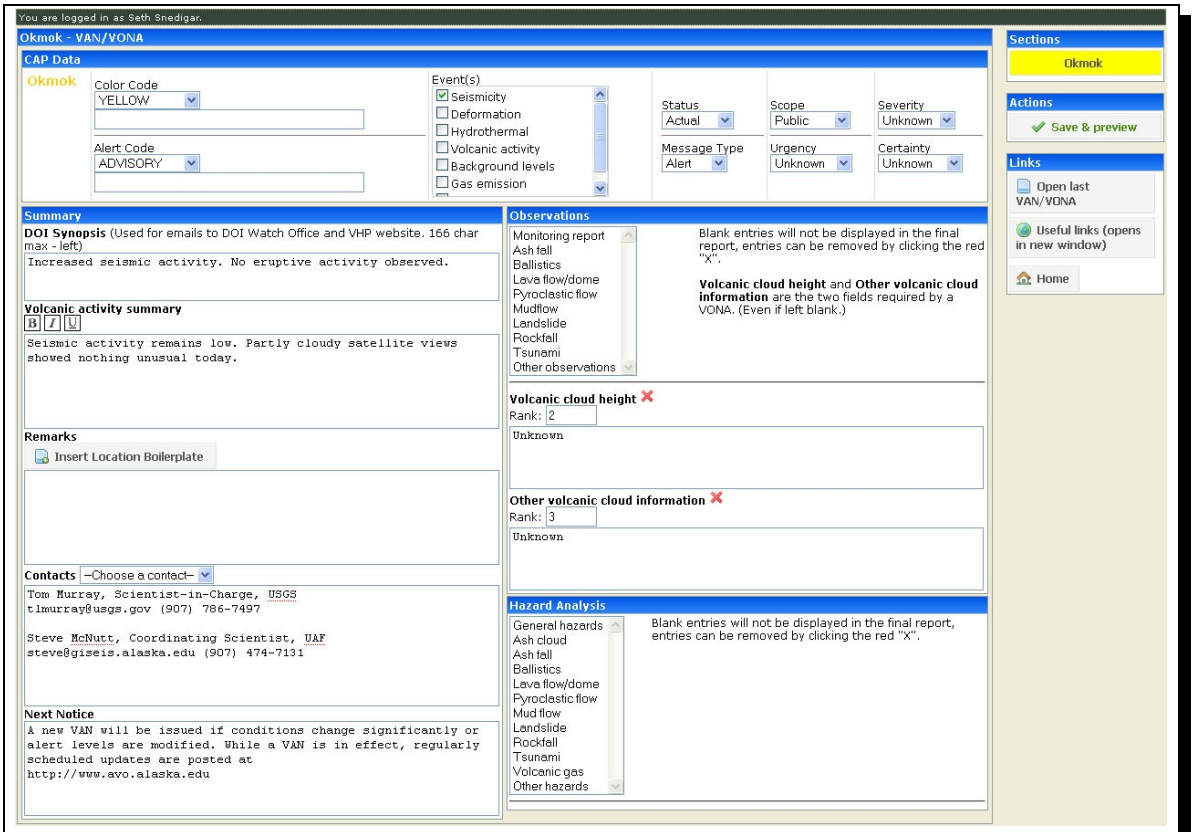

Figure 1: This figure shows the data entry module for a VAN/VONA for Okmok volcano.

As the user enters their data into the various forms for the associated volcanoes, they are able to choose an option to preview their release. This preview mode inserts all the data the user has entered in so far into a series of preview tables in the database. From there, the product format module pulls information from the preview tables and displays the report to the user as it exists at the time.

This report preview serves two purposes – one, it allows the user to see what their report looks like, and two, it saves the data for the user so they can logout of the system, and return to work on their report at a later time.

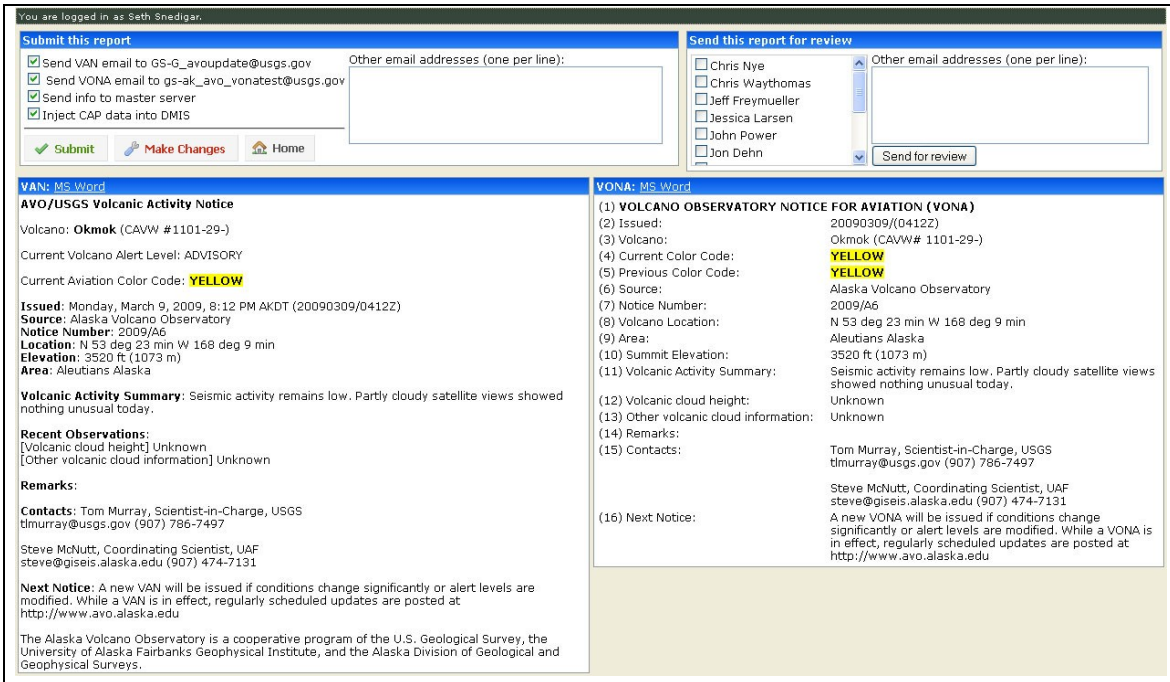

Figure 2: Report preview screen. User can look at their finished product so far, and go back to the editing form if needed. If the user is satisfied with the report, they can submit the report.

Finally, when the user is satisfied with their report, the data entry module enters the user's data into the production tables in the database, and notifies the product transmission module that a new release has been entered, and needs to be sent to the appropriate places.

Product Format Module

A wide range of formatted text products are created in HANS. The product format module takes in data entered by a user in the data entry module, and formats the data into the specific type of report requested by the user. This module must be flexible enough to accommodate new report formats that may be developed in the future.

Product Transmission Module

When a user submits a report, the data and the type of report created is sent on to the product transmission module. The transmission module then interacts with the product format module to create the necessary report types and send those reports on to the appropriate places.

For example, suppose a volcano is beginning to appear restless – the activity is starting to grow beyond background levels. If the observatory chooses to change the aviation color code and volcano alert level at the volcano, they will need to release a VAN and VONA. The scientist on duty at the time enters the appropriate information into the web forms, and submits the report. Once the report is submitted into the database, the transmission module calls the product format module and creates the formatted VAN product, VONA

product, and CAP product. The module then emails the VAN to the appropriate general hazards email list, emails the VONA to the appropriate aviation email subscription list, and inserts the CAP data into a queue destined for the DMIS servers. Finally, the transmission module turns the data over to the synchronization module, where it is sent to all other HANS instances.

Data Synchronization Module

One problem that has been nagging the VHP is fault tolerance. If one observatory's computer systems were unavailable for any reason, they would be unable to release their information except vocally over the phone. HANS v3.0 attempts to solve this problem through the data synchronization module and multiple instances. When a report is released, or a change made to information stored within HANS, the data synchronization module transmits the data to all other known installations of HANS. Data transmission is accomplished using the Simple Object Access Protocol (SOAP). The SOAP format and associated processes are described in the next section.

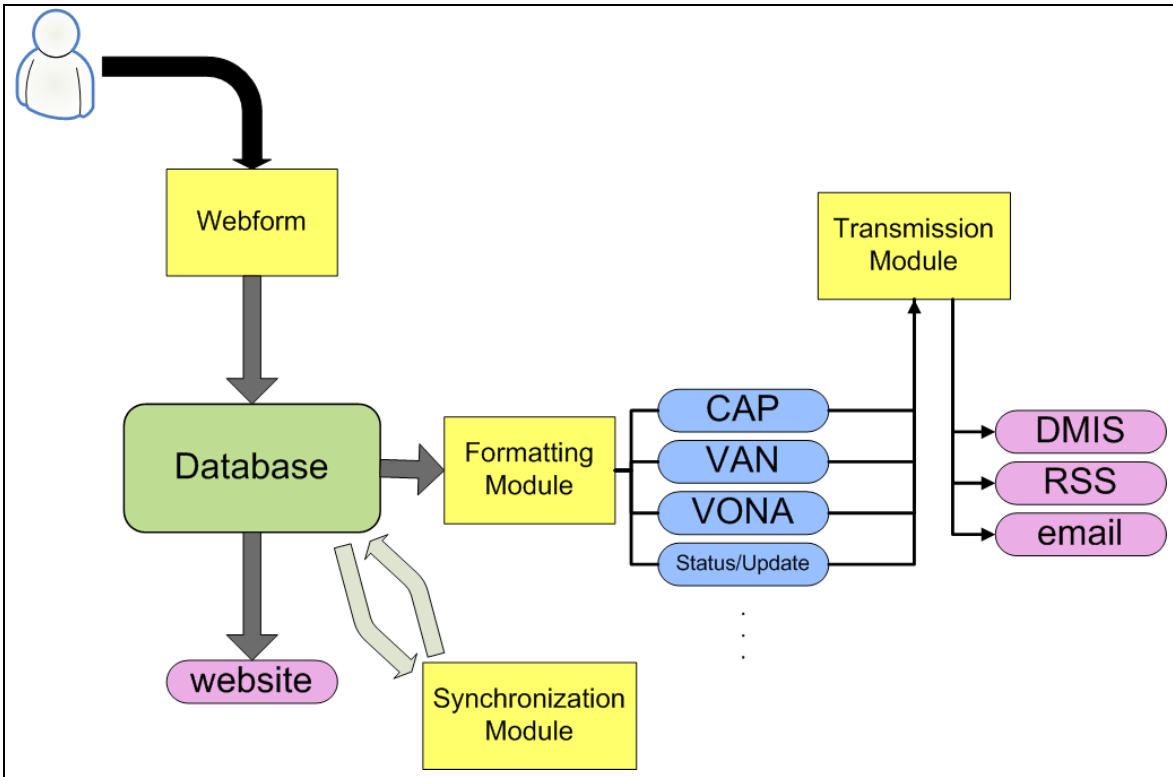

Figure 3: Logical view of the different modules and operations in HANS. The data entry module is located within the webform, where the user enters their data.

SOAP

SOAP [14] (originally known as Simple Object Access Protocol) is the protocol used for data transmission between HANS instances. SOAP is a simple, lightweight protocol for structured and strong-type information exchange in a decentralized, distributed environment [1]. The use of SOAP helps overcome a multitude of obstacles that would normally be seen in the network situation presented to HANS. Due to the interagency

nature of the volcano observatories, there are typically several firewalls and security layers between servers and the internet.

Although SOAP works well over a number of different web-type protocols (ftp, smtp, etc.), it is well suited to run over the primary web protocols – http and https (ports 80 and 443, respectively). Each observatory maintains its own web site, and thus has a possible entry into HANS. Using SOAP can help alleviate any problems that firewalls and agency policies may bring up [5].

Each observatory maintains their own computer systems, using their own update schedules and operating systems of choice. Due to the interoperable nature of SOAP, the operating system or database version does not particularly matter. Currently, each observatory uses MySQL as the database management system (DBMS). However, if an observatory used a different DBMS, the SOAP messaging would stay the same, but the code used to process the SOAP messages would need to be changed slightly to accommodate the different DBMS.

WSDL

The basis for SOAP operation is the Web Services Description Language (WSDL) [15]. (WSDL is not required to use SOAP, the protocol can be used on its own, but the WSDL can make operations, such as the ones seen in HANS much easier to define and use.) The WSDL file is an XML based document which describes services offered by a particular host machine. The WSDL file can be compared to the header file for a C++ class. Clients connecting to the host are able to discover what functionality that particular host offers. In HANS, each individual node can act as either a SOAP client, or a SOAP server, so each node offers identical services, from nearly identical WSDL files.

The WSDL file can be broken down into four sections: types – the data types used by the service, messages – the messages sent by the service, bindings – the communication protocols used by the service, and ports – the operations performed by the service.

Data Types

The data types are the native formats of the data encoded before the data is sent across the network to a receiving host. There are multiple methods of encoding data types; three different encodings were explored in developing HANS.

The first encoding method is a one to one mapping of fields within a particular database table to elements of a complex sequence in the WSDL file. Thus, a single complex element represents a single row in the table. SOAP libraries generally have error checking built in; using this type of data encoding guarantees that the data sent to or from a server will have exactly those fields specified in the WSDL file. Any extra fields added to the message, or fields missing from the message will cause the library to throw a SOAP exception.

One drawback to this encoding method is that it is tightly linked to the database. If any changes are made to the database schema, the WSDL file will need to be changed to

match the new schema. However, this limitation can be overcome by using a dynamically generated WSDL file, where the complex elements representing tables can be created by sing a SHOW COLUMNS query on the necessary tables.

The second data encoding method involves a simple sequence of two elements – a field name and field value – modeling a tuple in the database. When multiple sequences of tuples are combined as an array, a table in the database can easily be represented. This method is quite flexible, but leaves the error checking and handling up to the programmer.

The final data encoding method is the method primarily used in HANS. This method consists of a complex element made up of three string elements. The first string is the serialized representation of an object destined for a remote host. The second string simply states what type of object has been serialized. The third string consists of the host id of the server that is sending the data. This method is also quite flexible, and makes adding objects to be sent much easier. Rather than making changes to the WSDL file on all servers, the developer only needs to write code that generates a message, and code to receive and parse that generated message. However, this method relies on the scripting language chosen to serialize objects, forcing all servers to use the same language.

```
- <xsd:element name="generic_object">
  - <xsd:complexType>
    - <xsd:sequence>
        <xsd:element name="serialized_object" type="xsd:string" />
        <xsd:element name="object_type" type="xsd:string" />
        <xsd:element name="sender_id" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Figure 4: Generic object data type primarily used in HANS, as defined by the WSDL file.

When sending a single row from a table as an object, the encoding is fairly simple, the system only needs to retrieve that row, encode it as an object, then serialize the object. However, when a data object involves more than one table, as found when a table contains foreign keys to other tables. A single complex element may include all columns from a particular table, as well as arrays of columns from other table definitions. In the WSDL file, complex elements can be nested inside each other as far as needed to match the database schema. In HANS, the largest object consists of data from three separate tables. The function used to create the object representation and serialize the object pulls one row from the primary table, and all associated rows from linking tables.

Ports

The port type section of the WSDL file defines what operations or services are available on a particular host. There may be several operations defined for each data type in the WSDL file. In HANS, a maximum of four operations are defined for each data type:

two PUT operations, and two GET operations. A PUT operation describes a *push* of data to a SOAP server from a host, while a GET operation describes a *pull* of data from a SOAP server to a host.

```
- <wsdl:operation name="putObject">
    <wsdl:input message="tns:putObjectRequest" />
    <wsdl:output message="tns:putObjectResponse" />
  </wsdl:operation>
- <wsdl:operation name="putObjects">
    <wsdl:input message="tns:putObjectsRequest" />
    <wsdl:output message="tns:putObjectsResponse" />
  </wsdl:operation>
```

Figure 5: Two put operations defined in the HANS WSDL file. The putObject operation defines a push of a single object, while the putObjects operation defines the push of multiple objects.

HANS SOAP operations are quite similar in definition. Since the primary data encoding involves serialized objects, four operations are defined for this encoding. There are two operations for each data direction (to the server, and from the server). There is an operation for sending a single data object to a server, and retrieving a single data object from a server. Then there is an operation for sending multiple data objects at a time to a host, as well as retrieving multiple data objects at a time from a host.

Messages

Each port (operation or service) requires at least one message defining what kind of data should be sent or received from the server. Each individual message can contain many parts; these parts can be considered the "arguments" that are given to the SOAP operation mentioned above. These arguments can be any basic xml data type, or any data type defined in the "types" section of the WSDL file.

Generally speaking, an operation will have two messages: a request message, and a response message. The request message travels from the connecting host to the SOAP server, while the response message travels from the server to the connecting host. If a host is requesting data from a server, the request message typically contains a simple string describing what type of data is needed. The response message contains the data requested by the host.

```
- <wsdl:message name="getObjectRequest">
    <wsdl:part name="type" type="xsd:string" />
    <wsdl:part name="from" type="xsd:string" />
  </wsdl:message>
- <wsdl:message name="getObjectResponse">
    <wsdl:part name="serialized_data" type="xsd:string" />
  </wsdl:message>
- <wsdl:message name="getObjectsRequest">
    <wsdl:part name="type" type="xsd:string" />
    <wsdl:part name="from" type="xsd:string" />
  </wsdl:message>
- <wsdl:message name="getObjectsResponse">
    <wsdl:part name="serialized_string_array" type="tns:ArrayOfStrings" />
  </wsdl:message>
```

Figure 6: Messages defining a singular object retrieval (getObject), and a multiple object retrieval (getobjects).

Bindings

The final section in the WSDL file is the bindings section. This section describes where the SOAP service is located, and what protocols are used to transfer the data [15]. SOAP works with many protocols, as mentioned before, http(s) is used as the data transfer protocol, and rpc as the binding style.

DATA OBJECTS

Once the distinct data types that the SOAP service will transfer back and forth have been defined, code needs to be written to create and modify these objects representing these data types. HANS defines a class for each distinct object that will be transferred among HANS instances. Each class has four functions besides the *constructor* function – an *insert* function, *update* function, *duplicate* function, and *changed* function.

The *constructor* function simply creates a standard data object, with each database field associated with the object as a public accessible variable. If multiple tables are involved, the subtables associated with the primary table are created as array variables. If a database table key is passed to the constructor, the object is created with the data retrieved from the database matching the key. Otherwise, the object is created with all fields empty.

The *insert* function inserts new data into the relevant tables in the database. If an object requires data to be entered into multiple tables, the *insert* function must take into account the order in which data should be inserted. However, before the *insert* function proceeds with an y insert queries, it first runs the *duplicate* function.

The *duplicate* function merely checks to see if the receiving host already has the passed data in its own database. If a duplicate row is found in the database, the object attempts to update the data in its own database, by calling the *update* function.

The *update* function updates a row already in the database. If an object needs to store data in multiple tables, the update function must take into account all associated tables, and update them as necessary. However, before the update function proceeds, it calls the *changed* function.

The *changed* function checks to see if the data in the object received is newer than the data in the database. All objects stored in the database have a *last_update* field which holds the UTC timestamp of when that particular object was last updated. If the received data has a more recent timestamp, then the database is updated with the new data, and a successful update message is returned to the sender. If the data in the database is more recent, nothing is changed and a no change needed message is returned to the sender.

If any errors are reported during any of the object operations, a detailed error message is returned to the connecting host and logged in a file on the host on which the error occurred. The actual implementation of object functions can vary between data types, depending on the data involved. For example, a *duplicate* function may take many parameters to check if a row is already in the database, or it may only take a single id variable.

DATA TRANSFER

Actual data transfer within HANS occurs at scheduled intervals and at user request. Whenever a user makes a change to specific data in the system, or adds new data to the system, the user is presented with the option to update the master server, or, if the user is actually working on the master server, the option is given to update all slave machines.

Slave to Master

When users choose to edit specific information on a slave server (information specific to an observatory for example), they may want the edited information to propagate to all other HANS servers. The editing web form provides a checkbox that allows the user to choose if they want their submission to travel to the master server. After the form has been submitted, the slave server updates its own internal database. If the user has selected the option to transmit data to the master, the slave immediately attempts to connect to the master server. If the connection is successful, the slave then sends the changed data to the master, and reports back to the user on the success of the data transfer and subsequent data insertion into the master's database. If the connection is not successful – due to a network outage or any other matter – the slave server inserts the data into a queue for later transmission attempts.

Master to Slave

When the master receives a data transmission from a slave server, as mentioned above, it immediately attempts to update its own database. This process works as follows. During data transmission, the slave sends a serial representation of the data object, as well as the type of data object being sent. The master receives both strings of data, creates an empty object of the type being passed, unserializes the passed data, then attempts to insert the data using the empty object's *insert* function. This function checks for duplicate data, and inserts new data if no duplicates are found. If a duplicate is found, then a check is

made to see if the data has changed, based on UTC timestamps. If the data has changed, it is updated with the newly received data.

Once the master has finished processing the data, it returns a result message back to the sending slave server. Then, it attempts to send out the newly received data to all slaves (except the slave whence it received the data). For each slave in the master's database, the master attempts to send the data. If the transmission is successful, the master logs the message into its log file. If the master is unable to connect to the slave, it just inserts the data into its own queue for later transmission.

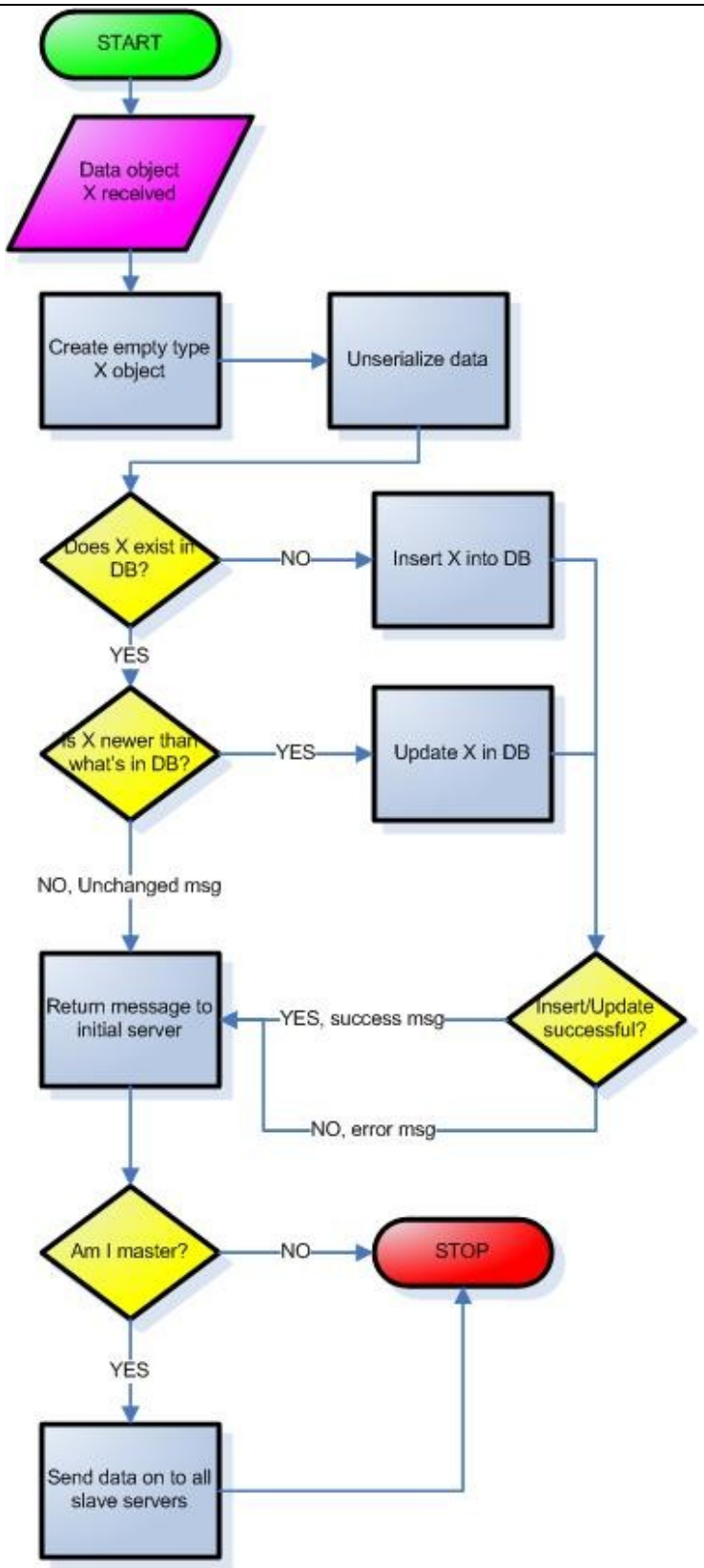

Figure 7: Flowchart of a servers processing as it receives a message from another server.

HANS uses the PHP scripting language to send and receive data via SOAP operations. One limitation of PHP is its inability to fork and spawn child processes, except in limited situations [9]. Since data reception, database updating, and data retransmission to slaves all occur within the same process, the master does not actually return any data to the slave until it has finished all database processing, and all subsequent sends of data to slaves. This means that the slave server could wait for quite a long time before a response is received from the master server. A forkable server side language that processes the SOAP reception and transmission could solve this issue; in HANS, PHP executes a system call which starts another php process running in the background. This background process executes the data transmission to all other slaves, so the primary process can return the appropriate message to the initial sending slave.

Data transfer performance

Performance is not typically an issue in HANS. Updates are sent infrequently, at most once or twice per hour. However, should the need arise to send data more frequently, studies have shown that SOAP can handle large amounts of data transferred over a short period of time [7].

Data transfer security

Security during data transfer in HANS has only been lightly considered, as the data is already considered public information, and there is no benefit to intercepting the data. However, a malicious user could possibly intercept the data stream, and change the contents of the data being transferred [10]. Also, there are web page passwords involved in data transmission that need to be kept encrypted. As such, HANS transmits all SOAP data through the secure socket layer (SSL). This allows data to be encrypted between the sending and receiving server.

DATA QUEUE

The queue is an integral part to the SOAP messaging process in HANS. If a message fails to reach a server, the sending server inserts that message into its queue for retransmission. The queue itself is a simple table in the database. Each row in the table stores a single serialized object string, the type of the object, the id of the host the object is destined for, and a timestamp marking the last attempt to send the object to the recipient.

Running the queue is a simple process. The host simply selects all data in the queue, ordered by destination. Then it simply goes through the data line by line, attempting to connect to the host and send the data. If the transmission is successful, the item is deleted from the queue. If not, the last attempt timestamp is updated, and the host machine tries again at a later time.

To speed up processing of the queue, the host running the queue creates a background process for each destination host in the queue. Thus it can transfer data concurrently, rather than in a serial manner. The queue is generally started by cronjob on the host

machine, running however often deemed necessary by the system administrator. It can also be started manually through the web interface to HANS.

DATABASE SCHEMA

Behind the scenes of HANS and integral to the entire operation of HANS is the database. There are currently twenty-four tables in the database; five tables are duplicate copies functioning as preview tables. Several tables contain background information used within HANS, but not necessarily used solely by HANS.

Background Tables

The *volcano* and *volcano_background* tables contain information about all volcanoes monitored by the observatories in the United States. HANS uses this information to describe particular volcanoes in reports by reporting elevation, latitude and longitude, and boilerplate description text. Information in these tables changes rarely, by other means besides HANS.

The *current_codes* table contains current volcano alert level and aviation color code data for all monitored volcanoes. The *previous_codes* table contains the previous volcano alert level and color code data for those same volcanoes. While this does not meet typical standard database normalization levels, these values are used throughout HANS and in several VHP webpages, so breaking the data up into these two different tables makes it much easier for other developers who may use the data.

Finally, *tblusers* contains user data for those users authenticated to use HANS. The table name is a hold-over from previous versions of HANS, as well as many other tools used internally to the observatories. Changing the name would require either storing duplicate information about usernames and passwords, or changing several other applications to accommodate the change. A new generic user login and authentication system is now being worked on within the VHP.

HANS Tables

The primary table storing release information is the *notice* table (and complementing *notice_preview* table). This table stores information unique to each notice or release. Several other tables store information associated to releases in the notice table. The *notice_text* (and *notice_text_preview*) tables store information that may be related to multiple releases. For example, AVO releases a Daily Status Report that contains information related to all volcanoes at elevated activity codes. Each volcano in the report has its own line in the notice table. However, each report often contains a header, footer, and contact information. It's redundant to store this information for each volcano, so it is stored separately in the *notice_text* table. Text stored in the *notice_text* table is linked to rows in the notice table through the *notice_linker* table (and *notice_linker_preview* table). One *notice_text* row can be linked to multiple *notice* rows.

Hazards and observations are both important parts to information releases, and required sections in the VAN and VONA. A list of possible hazards that can be reported is stored in the *hazards* table, and an observation list is stored in the *observations* table. Items can

be added to these tables as needed. Text about selected hazards or observations are stored in linking tables linking the text and hazard/observation with the associated row in the notice table. These linking tables are *notice_to_hazard* (and *notice_to_hazard_preview*), and *notice_to_observation* (and *notice_to_observation_preview*).

The *notice_contacts* table stores contact information for scientists who may be listed in an information release. This contact information is also used when sending out a notice for review. Some specific types of information releases are often asked to be reviewed by other scientists. HANS has the capability to send out emails to scientists listed in the contacts table, asking them to review a release. Any comments they post on this release are stored in the *notice_comments* table.

The *hans_links* table is a utility table. Scientists often add urls to other scientific sites in the release text. The *hans_links* table simply allows the scientists to store those urls so they can be easily accessed when entering information for a new release.

The *obs* table stores information about each of the observatories – the name, its associated acronym, email addresses, identifying information, etc.

The *hans_hosts* and *hans_queue* tables are related to the operations of the synchronization module. The *hans_hosts* table stores information about all the hosts that are currently running a HANS instance – ip addresses, username and password, WSDL file location, etc. The *hans_queue* table stores messages that were destined for a particular host, but were not able to reach that host for any particular reason. Each HANS instance attempts to send all messages in its own queue on a periodic basis.

The *notice_mail* table stored notice ids that have been released. This table will be used in a later system – the Volcano Notification System (VNS). This system will allow users to sign up to receive email notifications for activity at various volcanoes. The ids stored in this table will allow the VNS to decide if it is appropriate to send that report to a particular user.

Finally, the *tblcapsend* table stores notice ids destined for the DMIS system. This table is also a holdover from HANS v2.0. The code that injected data into the DMIS database was able to be ported to HANS v3.0 with very little modification; the table structured stayed exactly the same.

LOGGING

HANS logs nearly all user and system activity to three separate log files, allowing administrators to easily monitor and debug the system. The three log files each track a different part of the system. The *access log* tracks function calls and database requests, and the *error log* tracks all errors in the system, including interface errors and database (SQL) errors. Finally, the *soap log* tracks all SOAP messages transmitted by a host machine, as well as those messages received from other machines.

WORK AND DATA FLOW

There are several steps involved to creating and submitting a release of volcanic information. To begin, a scientist looks at all data available for the volcanoes he or she is interested in. These data streams typically include seismic and satellite data, and may also include GPS data, images from webcams, and pilot reports. The scientist must then synthesize all this data down into text readable and understandable by the general public. Once the scientist has chosen the appropriate product and entered the data into the necessary fields, they may choose to send the product off for review by other scientists in the VHP. Once the product has been reviewed, the scientist submits the data to the system. Here, the scientist's job is virtually finished, except for submitting a document to the fax system if necessary, or optionally checking to see if the correct web pages were updated.

From the data submission by the scientist, the data has its own distinctive flow through the system. If the scientist is submitted data to a slave server, the slave updates its own internal database, then sends the data on to the master. When the master receives the data from the slave, it updates its database, then transmits the data on to all slaves it knows about (except for the slave it just received data from). Conversely, if the scientist has submitted data directly to the master server, the master updates its own database, then transmits data to all slaves in the system.

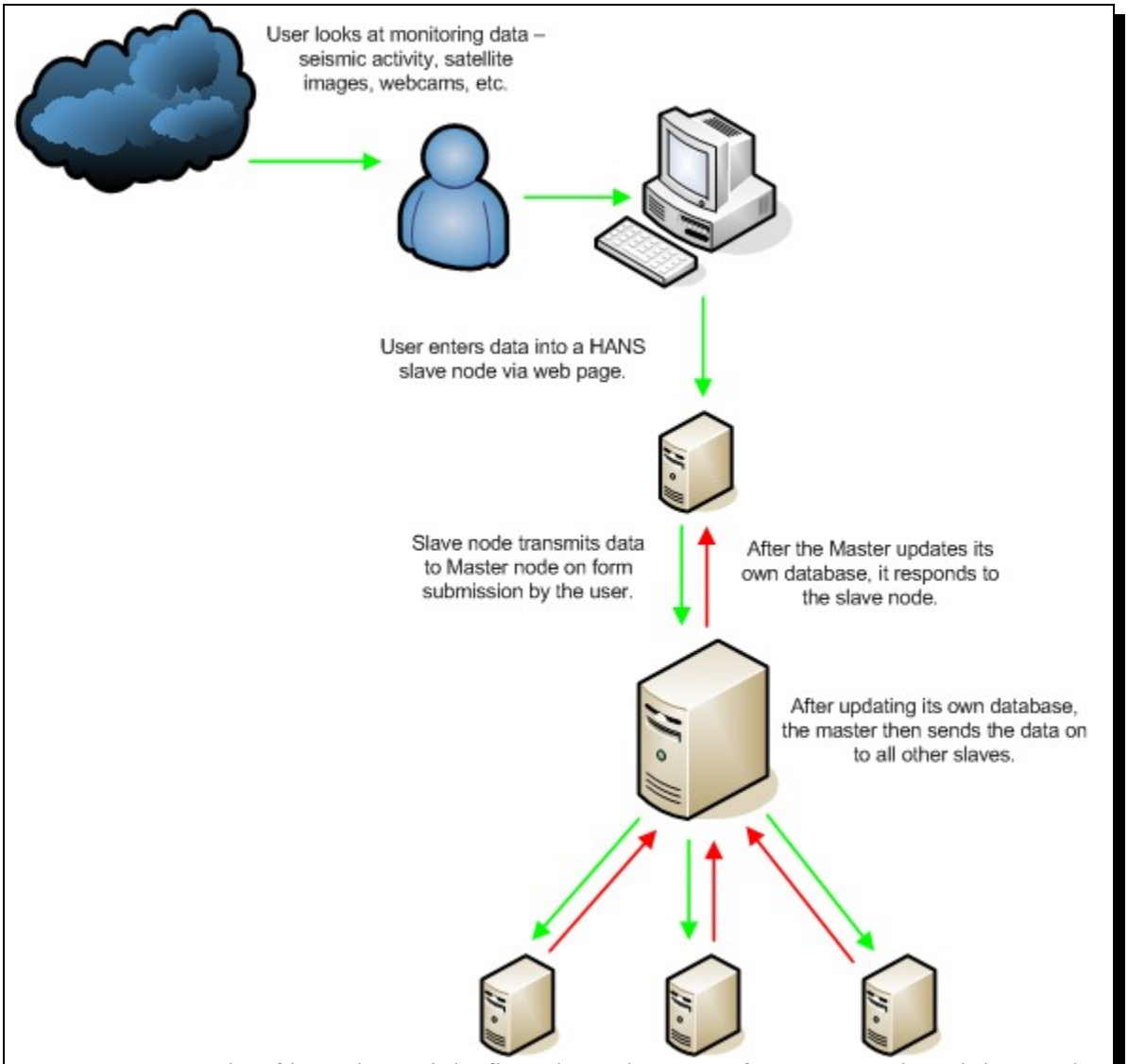

Figure 8: Example of how data might flow through HANS from user gathered data to the master server to all slave servers.

DEPLOYMENT

Many open source tools and software have been used and are still being used in the development and usage of HANS. Mentioned many times already is the Subversion source code repository application [12]. All code and database schemas are stored in the repository, allowing easy access to checkout the code, or to rollback to earlier versions if necessary.

As HANS is a web-based system, it requires a webserver to operate. Any webserver capable of running php code will serve the purpose of hosting HANS. The Apache http server is the most popular web server on the internet [13], and is currently used on all VHP machines to serve websites, as well as host HANS.

A database system is also needed for HANS to operate. MySQL is currently the world's most popular open source database [6]. The VHP has been using MySQL to serve their dynamic websites for several years; integrating HANS into these existing MySQL databases allows other programs to easily access data stored by HANS in the database.

Finally, the PHP scripting language is used to tie the database and webserver together, producing the interfaces used by the scientists to enter data, as well as synchronize the data between HANS instances. PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML [9].

Other secondary "languages" used in HANS include Structured Query Language (SQL), for sending data back and forth from the database, Hypertext Markup Language (HTML), for creating pages viewable by users using a web browser, Cascading Style Sheets (CSS), for styling said HTML pages, and Javascript, used to provide a more interactive web page to the user.

The choice to use the above open source software (OSS) packages was made for several reasons. A primary reason of course is the cost of the software (free). As many of the observatories are multi-agency operations (for example, AVO consists of personnel from three agencies: USGS, UAFGI, and ADGGS) the procurement procedure to purchase hardware and software is fairly complex. Using OSS products allows VHP developers to use the latest tools and products available without needing to spend large amounts of money on those tools and products. Also, due to the nature of OSS, the documentation and examples available for the tools used is widely available and easily used. Finally, those involved in the development of HANS are well experienced in the tools and languages used.

ADMINISTRATION

HANS is fairly easy to administer. In the VHP setting, one individual should be able to easily administer all HANS instances at all server locations – provided they are able to access all installations. Even assuming that all observatories have their own installation and backup installation, that brings the count up to ten installations, which is highly unlikely to occur.

Installation and configuration are the primary areas where administration is needed, although upgrades to the software may take a little time of an administrator's day.

Installation

All source code for HANS is currently stored in a Subversion repository located on AVO's webserver. Assuming the Subversion software is installed in a remote system, the recommended way to get the software is to simply check out the latest version from the repository. If Subversion is not installed on the destination machine, then the administrator can simply grab a zipped tarball of the source files from the AVO server.

Once the administrator has the source files either checked out or unzipped into a web accessible directory, the administrator can then install the database schema.

Database Installation

The HANS database installer rests in the install directory of the source code. An administrator can visit this directory in a web browser (we assume that the administrator has already placed the code in a password protected directory). The installer presents the administrator with a form allowing them to fill in a database host, username, password, and name. Once the installer has been armed with this information, it attempts to connect to the given database and create the necessary tables and fill them with installation data.

The source code contains a file that stores an SQL schema of all tables need by HANS, with some table being pre-filled with data (such as the volcano and volcano_background tables). The installation script looks at each table in the schema and checks if that table exists in the given database. If the table does exist in the given database, the installer then checks to ensure that table has all necessary fields. If any fields are missing, the installer adds them to the table. If the table does not exist in the database, the installer creates the table and adds any data needed to the table.

The installation script does not delete any tables or fields from tables, nor does it attempt to add any extra data to tables which already have data stored in them.

Configuration

Once the administrator has retrieved the source code and installed the database tables, they can move on to configuring the application. The configuration file is a simple php file; HANS comes with a sample file called hans.inc.sample.php. The administrator needs to copy this file to hans.inc.php, and make the necessary changes within the file. Each line in the file is documented as to what type of information should be entered.

Once the HANS configuration has been completed, the administrator will most likely want to add the current server into the network of other HANS instances. The administrator should know the information for the HANS master server, and will be able to enter this information into the Hosts section of the SOAP control panel. After adding the HANS master server to the hosts section, the administrator should be able to connect to the master and register the new host on the system, as well as being able to download the most current information about other hosts on the HANS network and the latest volcanic information transmitted by those hosts.

Upgrading

As with any software package, upgrades and bug fixes are an inevitable occurrence. As bug fixes and upgrades are added to HANS, they will need to be propagated out to all running HANS instances. All additions and changes to the software will be uploaded to the Subversion repository hosted by AVO. Changes and additions to the database schema will be made to the database at AVO, where the program to create the schema that is read by the HANS installer is stored. Once the changes to the schema have been made, the schema creator makes a new SQL schema file, where it can be added to the source code repository.

If an instance is running code that has been checked out from the repository, all the admin has to do is update to the latest version in the repository (simply issuing a svn update command). This will update the code, and then the administrator needs to run the database installer script. This will update and add any necessary fields and tables to the database.

CONCLUSION AND FUTURE WORK

HANS is used everyday – often multiple times a day during times of high volcanic activity – by VHP scientists. As such, suggestions are often made to increase the usability and functionality of the system. These suggestions are vetted for their usefulness and the complexity of adding them to the system. If the suggestion is deemed useful and not too complex, it is added to the system immediately, or added to the HANS TODO list.

Another project directly related to HANS is the Volcano Notification System (VNS). This project is in the early stages of research and requirements gathering. The VNS will allow individuals to customize the reports they want to receive via email from the various volcano observatories. The VNS will take reports generated by HANS, determine if they should be sent to the users signed up in VNS, and if appropriate, email the report to those users.

While email plays an important role in information transmission, all information releases are also faxed to several agencies that require information from the VHP. Currently, each observatory has its own system and method of faxing information. AVO uses an online fax system called Protus. While uploading a word document of the report to the fax system is not highly complex, an interface to a fax system would remove one more thing from the scientists to-do list.

Currently, any changes made to the product format module must be made directly to the code, and any new products that are created must be added to the code. A templating system added to HANS will allow users to create new reports directly from data in HANS without requiring a developer to make changes to the code first. The templating system would allow users to create new templates and store them in the database.

Finally, HANS could benefit from the addition of Web Service Security (WSS) [16] to the messaging system. Relying on SSL encryption is a first step towards security, but much could be done in the authentication of HANS instances with each other.

This paper has defined and identified what HANS is and how it works. The data synchronization used within HANS is a useful feature, and while it is not meant to replace true database replication, it does work to keep basic database systems updated with each other. The features developed in HANS will be useful as the VHP program moves towards more centralized applications, with each observatory hosting their own slightly customized version of these applications.

APPENDIX

Example VAN

AVO/USGS Volcanic Activity Notice

Volcano: **Okmok** (CAVW #1101-29-)

Current Volcano Alert Level: ADVISORY
Previous Volcano Alert Level: NORMAL

Current Aviation Color Code: **YELLOW**
Previous Aviation Color Code: **GREEN**

Issued: Monday, March 2, 2009, 8:52 PM AKST (20090302/0552Z)
Source: Alaska Volcano Observatory
Notice Number: 2009/A5
Location: N 53 deg 23 min W 168 deg 9 min
Elevation: 3520 ft (1073 m)
Area: Aleutians Alaska

Volcanic Activity Summary: AVO is increasing the Aviation Color Code to **YELLOW** and the Volcanic Alert Level to ADVISORY for Okmok volcano. Over the past 24 hours there have been short bursts of volcanic tremor at Okmok volcano. These bursts are occurring at a rate of one per hour on average. This is the first sign significant seismic activity at the volcano since the cessation of eruption in August 2008 and represents an increase above typical background activity.

This increase in seismic activity does not necessarily indicate that an eruption will occur, or that an eruption is likely. Clouds currently obscure satellite views of the volcano. However there is no evidence that an eruption has occurred. The AVO Operations Room is currently being staffed 24/7 due to unrest at Redoubt Volcano.

Recent Observations:
[Volcanic cloud height] None observed.
[Other volcanic cloud information] None observed.

Hazard Analysis:
[General hazards] Unstable, muddy surfaces and slopes of new volcanic debris within the caldera are prone to liquefaction and collapse. In addition, new lakes, ponds, and deep craters with steep, collapsing banks present hazards to anyone visiting the caldera. The interior of the new tephra cone was hot when imaged with an infrared camera in mid-September. Volcanic gas was not measured; however, noxious gases could be present around this new tephra cone. All drainages leading downslope from the rim of the caldera should be considered hazardous as ash and other loose debris may be remobilized suddenly by heavy rains. The Crater Creek drainage on the north-northeast flank of Okmok should be avoided as it may experience sudden flooding events if water impounded within the caldera breaches new tephra dams.

Remarks: Okmok volcano was in vigorous eruption from July 12 through mid-August, 2008. Energetic, intermittent ash emission from several vents within the caldera blanketed much of the northeast portion of Umnak Island with ash and dusted Unalaska/Dutch Harbor with fine ash on several occasions. Rain-induced remobilization of debris from high on the volcano's flanks produced lahars down many drainages, forming new deltas at the coastline. A new, 200-300 m-high (660-980 ft) tephra cone developed inside the caldera at the primary eruption site. Since late August 2008, seismicity at Okmok has generally declined. The last confirmed ash emission at Okmok occurred on August 19, 2008.

Okmok volcano is a 6-mile-wide caldera that occupies most of the eastern end of Umnak Island, located 75 miles southwest of Unalaska/Dutch Harbor in the eastern Aleutian Islands. Okmok has had several eruptions in historic time typically consisting of ash emissions occasionally to over 30,000 feet ASL but generally much lower; lava flows crossed the caldera floor in 1945 and 1958. Prior to 2008, the volcano last erupted in February 1997 producing lava flows and intermittent ash emissions over the course of several months.

The nearest settlement is Nikolski, population about 35, roughly 45 miles west of the volcano. A ranch caretaker family lives at Fort Glenn on the flank of the volcano about 6 miles east of the caldera rim.

Contacts: Tom Murray, Scientist-in-Charge, USGS
tlmurray@usgs.gov (907) 786-7497

Steve McNutt, Coordinating Scientist, UAF
steve@giseis.alaska.edu (907) 474-7131

Next Notice: A new VAN will be issued if conditions change significantly or alert levels are modified. While a VAN is in effect, regularly scheduled updates are posted at http://www.avo.alaska.edu

The Alaska Volcano Observatory is a cooperative program of the U.S. Geological Survey, the University of Alaska Fairbanks Geophysical Institute, and the Alaska Division of Geological and Geophysical Surveys.

Example VONA

(1) **VOLCANO OBSERVATORY NOTICE FOR AVIATION (VONA)**

(2) Issued:	20090302/(0552Z)
(3) Volcano:	Okmok (CAVW# 1101-29-)
(4) Current Color Code:	**YELLOW**
(5) Previous Color Code:	**GREEN**
(6) Source:	Alaska Volcano Observatory
(7) Notice Number:	2009/A5
(8) Volcano Location:	N 53 deg 23 min W 168 deg 9 min
(9) Area:	Aleutians Alaska
(10) Summit Elevation:	3520 ft (1073 m)
(11) Volcanic Activity Summary:	AVO is increasing the Aviation Color Code to **YELLOW** and the Volcanic Alert Level to ADVISORY for Okmok volcano. Over the past 24 hours there have been short bursts of volcanic tremor at Okmok volcano. These bursts are occurring at a rate of one per hour on average. This is the first sign significant seismic activity at the volcano since the cessation of eruption in August 2008 and represents an increase above typical background activity. This increase in seismic activity does not necessarily indicate that an eruption will occur, or that an eruption is likely. Clouds currently obscure satellite views of the volcano. However there is no evidence that an eruption has occurred. The AVO Operations Room is currently being staffed 24/7 due to unrest at Redoubt Volcano.
(12) Volcanic cloud height:	None observed.
(13) Other volcanic cloud information:	None observed.
(14) Remarks:	Okmok volcano was in vigorous eruption from July 12 through mid-August, 2008. Energetic, intermittent ash emission from several vents within the caldera blanketed much of the northeast portion of Umnak Island with ash and dusted Unalaska/Dutch Harbor with fine ash on several occasions. Rain-induced remobilization of debris from high on the volcano's flanks produced lahars down many drainages, forming new deltas at the coastline. A new, 200-300 m-high (660-980 ft) tephra cone developed inside the caldera at the primary eruption site. Since late August 2008, seismicity at Okmok has generally declined. The last confirmed ash emission at Okmok occurred on August 19, 2008.

Okmok volcano is a 6-mile-wide caldera that occupies most of the eastern end of Umnak Island, located 75 miles southwest of Unalaska/Dutch Harbor in the eastern Aleutian Islands. Okmok has had several eruptions in historic time typically consisting of ash emissions occasionally to over 30,000 feet ASL but generally much lower; lava flows crossed the caldera floor in 1945 and 1958. Prior to 2008, the volcano last erupted in February 1997 producing lava flows and intermittent ash emissions over the course of several months.

The nearest settlement is Nikolski, population about 35, roughly 45 miles west of the volcano. A ranch caretaker family lives at Fort Glenn on the flank of the volcano about 6 miles east of the caldera rim.

(15) Contacts:

Tom Murray, Scientist-in-Charge, USGS
tlmurray@usgs.gov (907) 786-7497

Steve McNutt, Coordinating Scientist, UAF
steve@giseis.alaska.edu (907) 474-7131

(16) Next Notice:

A new VONA will be issued if conditions change significantly or alert levels are modified. While a VONA is in effect, regularly scheduled updates are posted at http://www.avo.alaska.edu

REFERENCES

[1] Cenov, M. 2003. WAN Communication using SOAP protocol. In *Proceedings of the 4th international Conference Conference on Computer Systems and Technologies: E-Learning* (Rousse, Bulgaria, June 19 - 20, 2003). B. Rachev and A. Smrikarov, Eds. CompSysTech '03. ACM, New York, NY, 406-410. DOI= http://doi.acm.org/10.1145/973620.973688

[2] Common Alerting Protocol (CAP), 1.1, [http://www.oasis-open.org/specs/#capv1.1] 2005.

[3] Gardner, C. A., Guffanti, M. C., U.S. Geological Survey's Alert Notification System for Volcanic Activity: U.S. Geological Survey Fact Sheet 2006-3139, 4p. [http://pubs.usgs.gov/fs/2006/3139].

[4] Guffanti, M., Brantley, S. R., Cervelli, P. F., Nye, C. J., Serafino, G. N., Siebert, L., Venezky, D. Y., and Wald, L., 2007, Technical-information products for a National Volcano Early Warning System: U.S. Geological Survey Open File Report 2007-1250, 23p. [http://pubs.usgs.gov/of/2007/1250].

[5] Jepsen, T., "SOAP cleans up interoperability problems on the Web," *IT Professional* , vol.3, no.1, pp.52-55, Jan/Feb 2001

[6] MySQL Database software, http://www.mysql.com, 2008

[7] Pasteur, O.; Tuan Dang; Delon, P.-E., "Using Web Services to exchange power plant process data," *Industrial Informatics, 2007 5th IEEE International Conference on* , vol.1, no., pp.411-415, 23-27 June 2007

[8] Pautasso, C., Zimmermann, O., and Leymann, F. 2008. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceeding of the 17th international Conference on World Wide Web* (Beijing, China, April 21 - 25, 2008). WWW '08. ACM, New York, NY, 805-814. DOI= http://doi.acm.org/10.1145/1367497.1367606

[9] PHP (Hypertext Pre-Processor) scripting language, http://www.php.net, 2008

[10] Rahaman, M. A., Schaad, A., and Rits, M. 2006. Towards secure SOAP message exchange in a SOA. In *Proceedings of the 3rd ACM Workshop on Secure Web Services* (Alexandria, Virginia, USA, November 03 - 03, 2006). SWS '06. ACM, New York, NY, 77-84. DOI= http://doi.acm.org/10.1145/1180367.1180382

[11] Robert T. Stafford Disaster Relief and Emergency Assistance Act, P.L 93-288, http://www.fema.gov/about/stafact.shtm, 2007

[12] Subversion version control system, http://subversion.tigris.org, 2008

[13] The Apache HTTP Server Project, http://httpd.apache.org, 2008

[14] W3C, Simple Object Access Protocol (SOAP), 1.2, http://www.w3.org/TR/soap/, 2007.

[15] W3C, Web Services Description Language (WSDL), 1.1, http://www.w3.org/TR/wsdl, 2001

[16] WS-Security, Web Services Security, 2004 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf